Capitolo 2: Automi a stati finiti

1 Sistemi a stati

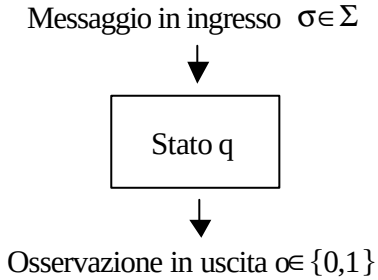
Vogliamo introdurre un sistema che modelli un semplice meccanismo di interazione con una "scatola nera", dotata di un ingresso e di una uscita. Supponiamo che tale sistema possa ricevere in ingresso messaggi, dati da un alfabeto finito $S = \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$, e produrre in uscita un valore in $\{0,1\}$che può essere osservato. Un *esperimento* sul sistema consiste nei due passi:

1. Viene presentata una sequenza di messaggi, descritta da una parola $w \in \Sigma^*$.
2. Al termine, viene effettuata una osservazione: se il risultato è 1 la parola viene accettata, altrimenti respinta.

Poiché possiamo accedere al sistema solo attraverso esperimenti, il *comportamento* del sistema è descritto dall'insieme di parole accettate; questo sistema è quindi visto come *riconoscitore* di linguaggi.

Esso può essere modellato attribuendo al sistema un insieme di possibili stati interni Q, con le seguenti richieste:

1. Ad ogni istante il sistema si trova in un preciso stato $q \in Q$ e questo stato può essere modificato solo attraverso un messaggio $\sigma \in \Sigma$ inviato in ingresso. La legge che descrive la modifica di stato interno causata dall'arrivo di un messaggio è data dalla *funzione di transizione* (o di *stato prossimo*) δ: $\Sigma x Q \to Q$. Se il sistema si trova nello stato q ed arriva il messaggio σ, il sistema passa nello stato $\delta(q, \sigma)$.
2. Prima dell'arrivo di ogni messaggio, il sistema si trova in uno stato iniziale $q_o \in Q$.
3. L'osservazione sul sistema è descritta da una *funzione di uscita* λ: $Q \to \{0,1\}$: se il sistema è nello stato q, il risultato dell'osservazione è $\lambda(q)$. Si osservi che la funzione λ è univocamente individuata assegnando l'insieme di stati $F = \{q \mid \lambda(q) = 1\}$; tali stati sono detti *stati finali*.



Formalmente:

Definizione 1.1 Un *automa a stati* A è un sistema $A = < Q , \Sigma , \delta , q_o , F >$, dove Q è *un insieme di stati,* Σ è un alfabeto finito, δ: $\Sigma x Q \to Q$ è la *funzione di transizione,* $q_o \in Q$ è lo stato iniziale, $F \subseteq Q$ è l'insieme degli stati finali che definisce una funzione λ: $Q \to \{0,1\}$, dove:

$\lambda(q)$ = se $q \in F$ allora 1 altrimenti 0

Se l'insieme Q è finito, l'automa è detto *a stati finiti.*

La funzione δ: $\Sigma xQ \to Q$ può essere estesa univocamente ad una funzione δ^*: $\Sigma^* xQ \to Q$ definita induttivamente da:

1. $\delta^*(q, \varepsilon)=q$,
2. $\delta^*(q, w\sigma)= \delta(\delta^*(w,q), \sigma)$ per ogni $w \in \Sigma^*$.

In sostanza $\delta^*(q, w)$ è lo stato in cui arriva il sistema, inizializzato con q, dopo aver applicato le azioni corrispondenti alla sequenza w di messaggi.

Il linguaggio L(A) riconosciuto dall'automa A è dato da:

$$L(A) = \{w \mid w \in \Sigma^* \text{ e } \delta^*(q_o,w) \in F \} = \{w \mid w \in \Sigma^* \text{ e } \lambda(\delta^*(q_o, w))=1 \}$$

Con abuso di notazione, per semplicità, la funzione δ^* sarà in seguito denotata con δ.

Un automa a stati finiti può essere ulteriormente rappresentato come *diagramma degli stati*, cioè come un grafo orientato in cui i vertici (indicati in circonferenze) rappresentano gli stati e i lati (etichettati con simboli di S) le possibili transizioni tra stati. Lo stato iniziale viene indicato con una freccia in ingresso mentre gli stati finali vengono indicati con una doppia circonferenza (vedere esempio 2.1).

Esempio 2.1

Si consideri l'automa A= < S, Q, d, q_0, F> dove:

S={a,b}

Q={q_1, q_2}

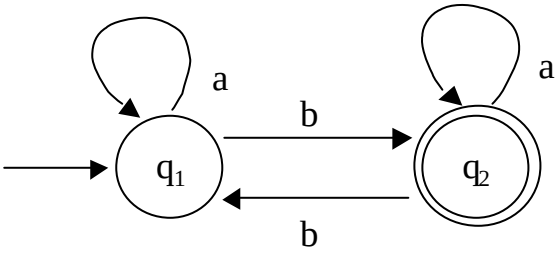
d: {a,b}x{ q_1, q_2}$\to$ { q_1, q_2} data dalla tabella :

d	a	b
q_1	q_1	q_2
q_2	q_2	q_1

Stato iniziale: q_1

F={q_2}

Il suo diagramma a stati è il seguente:



Si osservi che ogni parola $w \in \Sigma^*$ induce nel diagramma degli stati un cammino, dallo stato iniziale q_o ad uno stato q, così che il linguaggio accettato dall'automa è dato dalle parole che inducono cammini che portano dallo stato iniziale in uno stato finale.

Nel corso di questo capitolo studiamo gli automi come riconoscitori di linguaggi. In particolare affrontiamo problemi di sintesi (dato un linguaggio, costruire un automa che lo riconosce) e di sintesi ottima (dato un linguaggio, costruire il "più piccolo" automa che lo riconosce). Diamo poi due caratterizzazioni della classe dei linguaggi riconosciuta da automi a stati finiti, la prima mediante le grammatiche di tipo 3 e la seconda mediante le cosidette "espressioni regolari".

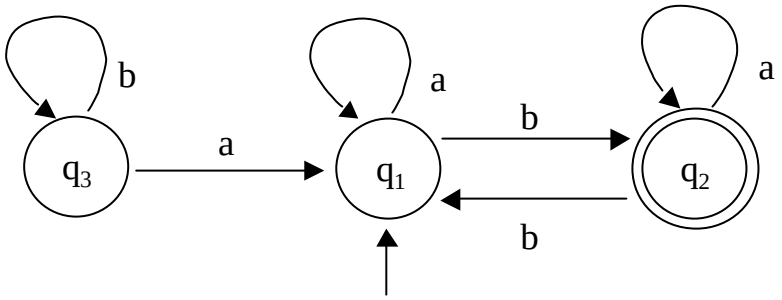
2 Osservabilità e indistinguibilità di stati

Dato un automa a stati $A = < Q , \Sigma , \delta , q_o , F >$, il comportamento di A è il linguaggio

$$L(A) = \{w \mid w \in \Sigma^* \text{ e } \delta(q_o, w) \in F \} = \{w \mid w \in \Sigma^* \text{ e } \lambda(\delta(q_o, w))=1 \}$$

Essenzialmente il comportamento è ottenuto dai risultati degli esperimenti sull'automa, poiché $\lambda(\delta(q_o, w))$ è il risultato di una osservazione dopo che l'automa ha processato la sequenza di messaggi descritta dalla parola w. Uno stato $q \in Q$ è detto *osservabile* se esiste una parola $w \in \Sigma^*$ per cui $q = \delta(q_o, w)$; un automa A è detto *osservabile* se tutti i suoi stati sono osservabili.

Esempio 2.2
Nel seguente automa lo stato q_3 è irraggiungibile o non osservabile, mentre q_1 e q_2 sono osservabili:



Gli stati non osservabili in un automa sono irrilevanti per quanto riguarda il riconoscimento e possono essere tranquillamente soppressi rendendo l'automa osservabile: per questo motivo in seguito supponiamo di trattare solo automi osservabili.

Dato un automa $A = < Q , \Sigma , \delta , q_o , F >$, due stati $q , q' \in Q$ sono detti *distinguibili* se si può trovare una parola $w \in \Sigma^*$ per cui $\lambda(\delta(q, w)) \neq \lambda(\delta(q', w))$: è cioè possibile distinguere con un esperimento il comportamento dell'automa inizializzato con q da quello inizializzato con q'. Se viceversa per ogni parola $w \in \Sigma^*$ vale che $\lambda(\delta(q, w))= \lambda(\delta(q', w))$, diremo q e q' che sono *indistinguibili*, scrivendo $q \approx q'$. Questo significa che non è possibile distinguere con esperimenti il comportamento dell'automa inizializzato con q da quello inizializzato con q'. Ovviamente due stati sono indistinguibili se e solo se non sono distinguibili.

Proposizione 2.1 $\approx$ è una relazione di equivalenza, verifica cioè le proprietà:
1. Riflessiva: $\forall q \ (q \approx q)$
2. Simmetrica: $\forall q,q' \ (q \approx q' \Rightarrow q' \approx q)$
3. Transitiva: $\forall q,q',q'' \ (q \approx q' \text{ e } q' \approx q'' \Rightarrow q \approx q'')$.

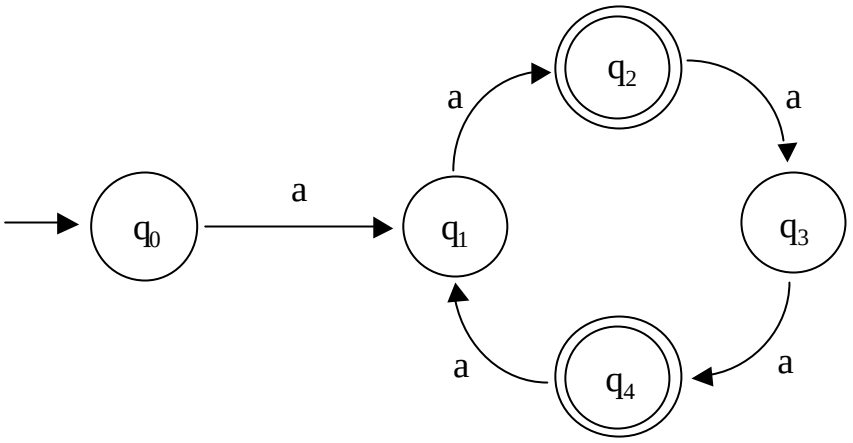
$\approx$ verifica inoltre la proprietà:

4. Se $q \approx q'$, allora $\delta(q, z) \approx \delta(q', z)$ per ogni $z \in \Sigma^*$.

Dimostrazione: E' immediato dalla definizione mostrare che $\approx$ è una relazione di equivalenza. Verifichiamo quindi solo la proprietà 4. Supponiamo che $q \approx q'$. Per parole $w,x \in \Sigma^*$ vale che $\delta(\delta(x, q), w)= \delta(q, wx)$ e $\delta(\delta(x, q',w))= \delta(q', wx)$; ne segue che per ogni parola $w \in \Sigma^*$ vale $\lambda(\delta(\delta(q,x),w))=\lambda(\delta(q, wx))= \lambda(\delta(q', wx))= \lambda(\delta(\delta(q',x),w))$, poiché $q \approx q'$. Questo prova che $\delta(q,z)) \approx (\delta(q',z)$.

ÿ

Esempio 2.3
Si consideri il seguente automa, rappresentato dal diagramma degli stati:



In tale automa q_1 e q_3 sono indistinguibili, q_2 e q_4 sono indistinguibili mentre sono tutti gli stati q_0 , q_1 e q_2 sono distinguibili tra loro.

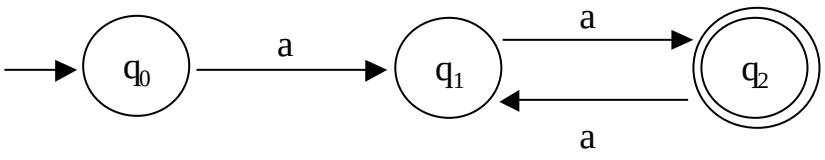
La presenza in un automa A di stati indistinguibili ci permette di costruire un nuovo automa $A_{\approx}$, ottenuto da A "identificando" gli stati indistinguibili tra loro. $A_{\approx}$ ha quindi "meno" stati di A pur riconoscendo lo stesso linguaggio.

Più precisamente, osserviamo che la relazione $\approx$ partiziona l'insieme degli stati Q in classi di equivalenza; indichiamo con $[q]_{\approx}$ la classe di equivalenza contenente q, cioè l'insieme di stati q' con $q \approx q'$. Possiamo ora costruire un nuovo automa $A_{\approx}$ che ha come stati le classi di equivalenza $[q]_{\approx}$, come funzione di transizione $\delta([q]_{\approx}, \sigma) = [\delta(q, \sigma)]_{\approx}$, come stato iniziale la classe di equivalenza $[q_o]_{\approx}$, come funzione λ la funzione $\lambda([q]_{\approx}) = \lambda(q)$.

Si osservi che, a causa della Proposizione 2.1, tali definizioni sono ben poste e che il nuovo automa riconosce lo stesso linguaggio del precedente: $L(A)=L(A_{\approx})$.

Esempio 2.4
Riprendendo **Esempio 2.3**, si ottiene l'automa $A_{\approx}$ dove:

osservabili:



3 Sintesi di automi

Affrontiamo in questa sezione il problema di sintesi di automi: dato un linguaggio $L \subseteq \Sigma^*$, costruire un automa che riconosca L. Considereremo in particolare *il più grande* automa osservabile che riconosce L e il *minimo* automa che riconosce L.

Osserviamo che un automa A associa ad ogni parola $w \in \Sigma^*$ lo stato $\delta(q_o, w)$, definendo dunque una funzione f: $\Sigma^* \to Q$ dove $f(w) = \delta(q_o, w)$. In un automa osservabile, per ogni stato q esiste una parola $w \in \Sigma^*$ tale che $q = \delta(q_o, w)$: la funzione f risulta quindi essere suriettiva.

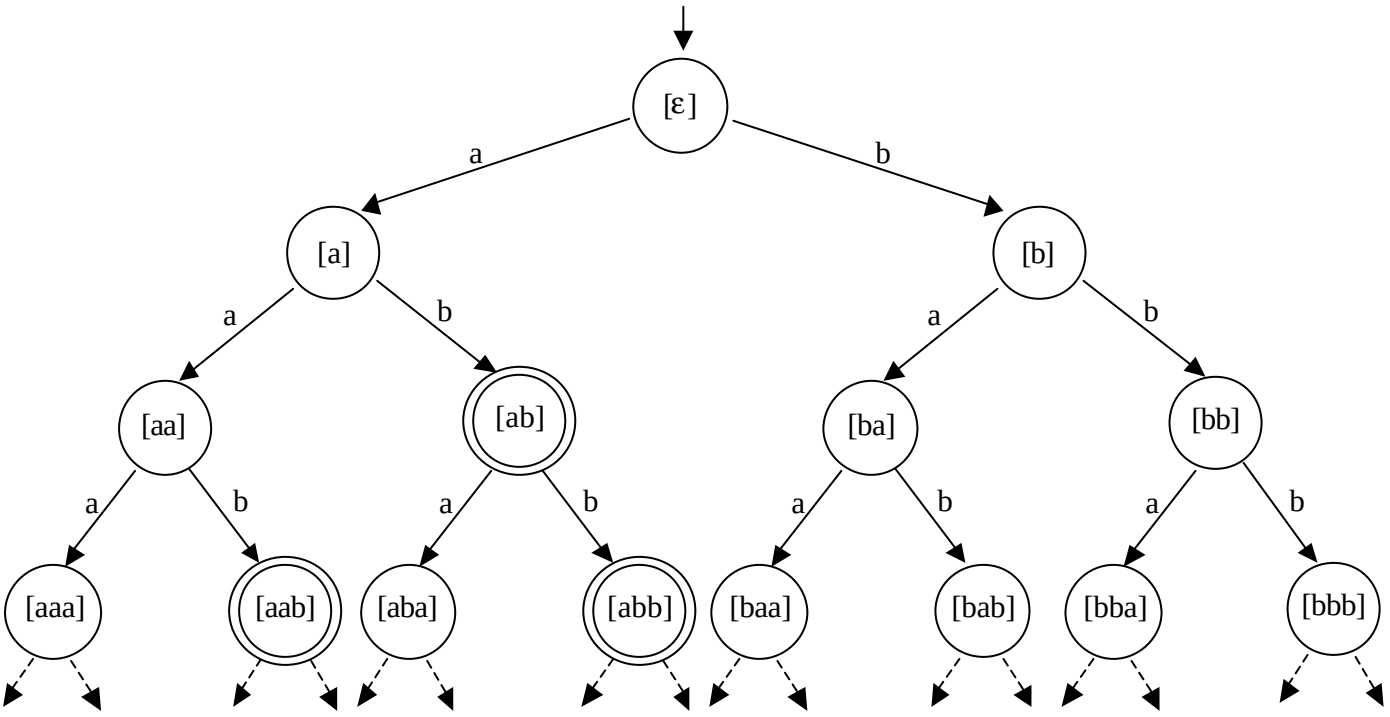
Nel *più grande* automa osservabile per un linguaggio L parole diverse corrispondono a stati diversi, cioè se $w \neq w'$ deve essere $\delta(q_o, w) \neq \delta(q_o, w')$. La funzione f: $\Sigma^* \to Q$ risulta allora una corrispondenza biunivoca: indicheremo con [w] lo stato corrispondente alla parola w.

Dato un linguaggio $I \subseteq \Sigma^*$, il *più grande* automa osservabile per L è l'automa $G_L = < Q , \Sigma , \delta , q_o , F >$ dove:

1. $Q = \{[x] \mid x \in \Sigma^*\}$
2. $\delta([x], \sigma) = [x\sigma]$
3. $q_o = [\varepsilon]$
4. $F = \{[y] \mid y \in L\}$

Si osservi che gli stati di G_L sono essenzialmente le parole in Σ^*, quindi per qualsiasi L l'automa G_L ha sempre infiniti stati.

Esempio 3.1
Il grafo degli stati G_L per il linguaggio $L = \{a^n b^m \mid m,n>0\}$ è il seguente:



Tornando al caso generale, osserviamo che due stati [x], [y] in G_L sono indistinguibili se, per ogni w, $xw \in L \Leftrightarrow yw \in L$. L'automa minimo M_L per L è ottenuto identificando gli stati indistinguibili in G_L, con la costruzione presentata in Sezione 2. Come ci si può aspettare, l'automa minimo è, fra tutti gli automi che riconoscono lo stesso linguaggio, quello che ha il minimo numero di stati.

Proposizione 3.1 Se $A = < Q , \Sigma , \delta , q_o , F >$ è un automa che riconosce L, il numero di stati di M_L non può superare quello di A.
Dimostrazione: Siano [x], [y] due stati distinguibili in G_L, e quindi per una opportuna parola w vale ad esempio che $xw \in L$ ma $yw \notin L$. Allora gli stati $\delta(q_o, x)$ e $\delta(q_o, y)$ sono stati di A distinti, poiché $\lambda(\delta(\delta(q_o, x), w) = \lambda(\delta(q_o, xw)) = 1$, ma $\lambda(\delta(\delta(q_o, y), w) = \lambda(\delta(q_o, yw)) = 0$.
Siano ora $[x_1]_{\approx}$, $[x_2]_{\approx}$, $[x_3]_{\approx}$, … tutti gli stati di M_L ; poiché $[x_1]$, $[x_2]$, $[x_3]$, … sono stati distinguibili in G_L , per quanto detto sopra $\delta(q_o, x_1)$, $\delta(q_o, x_2)$, $\delta(q_o, x_3)$, … sono stati distinti in A. A possiede almeno tanti stati quanto M_L , da cui la tesi.

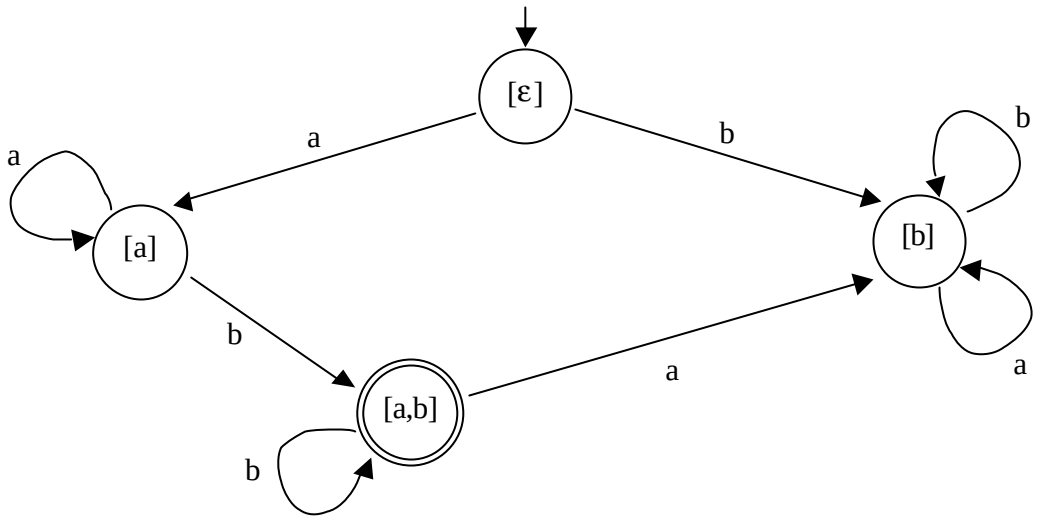
ÿ

L'automa minimo per un linguaggio L può essere dunque determinato dal seguente procedimento:

1. Considera l'automa G_L (descritto da un albero con infiniti nodi)
2. Partendo dalla radice [ε], visita l'albero in ampiezza e modificalo, fino a quando ci sono nuovi nodi da visitare. Visitando il nodo [wσ], cui arriva un arco etichettato σ dal nodo [w], se [wσ] è indistinguibile da un nodo [x] già precedentemente visitato allora collega [w] a [x] con un arco etichettato σ e cancella il sottoalbero di radice [wσ].
3. Il nodo iniziale del grafo degli stati di è [ε], i nodi finali sono quelli del tipo [x] con $x \in L$.

Esempio 3.2
Sia $L=\{a^n b^m | \ m,n>0\}$.

Per costruire M_L si parta a visitare in ampiezza G_L dalla radice [ε].
1. [a] è distinguibile da [ε], poiché $\varepsilon b \notin L$ ma $ab \in L$.
2. [b] è distinguibile sia da [ε] che da [a], poiché $bab \notin L$ ma $\varepsilon ab \in L$ e $bb \notin L$ ma $ab \in L$.
3. [aa] è indistinguibile da [a], come si verifica facilmente. Collega allora [a] a se stesso con un arco etichettato "a" e sopprimi il sottoalbero di radice [aa].
4. [ab] è distinguibile da [ε], da [a] e da [b], come si verifica facilmente (per esempio, [ab] è distinguibile da [b] perché $bb \notin L$ ma $abb \in L$).
5. [ba] è indistinguibile da [b], come si verifica facilmente. Collega allora [b] a se stesso con un arco etichettato "a" e sopprimi il sottoalbero di radice [ba].
6. [bb] è indistinguibile da [b], come si verifica facilmente. Collega allora [b] a se stesso con un arco etichettato "b" e sopprimi il sottoalbero di radice [bb].
7. [aba] è indistinguibile da [b], come si verifica facilmente. Collega allora [ab] a [b] con un arco etichettato "a" e sopprimi il sottoalbero di radice [aba].
8. [abb] è indistinguibile da [ab], come si verifica facilmente. Collega allora [ab] a se stesso con un arco etichettato "b" e sopprimi il sottoalbero di radice [aba].
9. Non esistendo nuovi nodi da visitare, la costruzione è terminata. L'automa minimo per $L=\{a^n b^m | \ m,n>0\}$ è allora descritto dal seguente grafo degli stati:



Esempio 3.3
Sia $L=\{a^n b^n \mid n>0\}$. Osserviamo che, in G_L , per $k,n>0$ e se $k \neq n$ vale che $[a^k]$ è distinguibile da $[a^n]$, poichè $a^n b^n \in L$ ma $a^k b^n \notin L$. Ne segue che in M_L gli stati $[a]_{\approx}$, $[a^2]_{\approx}$, … , $[a^k]_{\approx}$, … sono distinti: l'automa minimo contiene dunque infiniti stati, quindi il linguaggio $L=\{a^n b^n \mid n>0\}$ non può essere riconosciuto da automi a stati finiti.

4 Automi a stati finiti e grammatiche di tipo 3

In questa sezione proviamo che i linguaggi regolari (di tipo 3) sono esattamente quelli riconosciuti da automi a stati finiti; gli automi a stati finiti risultano dunque i riconoscitori corrispondenti a quei sistemi generativi che sono le grammatiche di tipo 3.

Proposizione 4.1 Per ogni linguaggio L riconosciuto da un automa a stati finiti esiste una grammatica G di tipo 3 tale che $L=L_G$.
Dimostrazione: Sia $A = < Q , \Sigma , \delta , q_o , F >$ un automa a stati finiti che riconosce L. Dato A, costruiamo la seguente grammatica $G=< S,Q,P, q_o >$ dove:
1. L'insieme dei metasimboli coincide con gli stati dell'automa.
2. L'assioma è lo stato iniziale.
3. $q_k \rightarrow \sigma q_j$ è una regola di produzione di G se $q_j = \delta(q_k, \sigma)$.
4. $q_k \rightarrow \varepsilon$ è una regola di produzione di G se $q_k \in F$.

Per induzione sulla lunghezza della parola, si prova che:

$$(1) \quad q_j = \delta(q_0, w) \text{ se e solo se } q_0 \Rightarrow_G^* w\, q_j \text{ per ogni parola } w$$

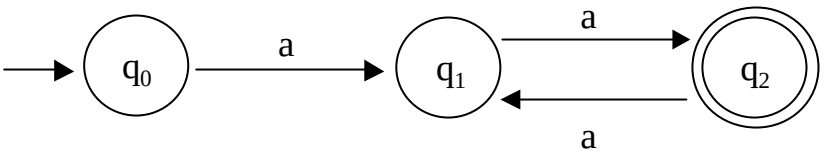
Se $l(w)=0$, cioè $w=\varepsilon$, la proprietà (1) è verificata. Supponiamo che la proprietà sia verificata per tutte le parole di lunghezza al più n, e consideriamo una parola $w\sigma$ di lunghezza n+1. Supponiamo che $q_j=\delta(q_0, w)$ e che $q= \delta(q_0, w\sigma)$, così che $q= \delta(q_j, \sigma)$ poiché $\delta(q_0, w\sigma)= \delta(\delta(q_0, w), \sigma)$,Poiché $l(w)=n$, per ipotesi di induzione esiste un unico q_j tale che $q_0 \Rightarrow_G^* w\, q_j$; per la regola di costruzione della grammatica numero 3, l'unica regola in G che permette di riscrivere q_j con una parola iniziante per q è $q_j \rightarrow \sigma q$. Ne segue:

$$q = \delta(q_0, w\sigma) \text{ se e solo se } q_0 \Rightarrow_G^* w\sigma q$$

Se q_j è stato finale, allora osserviamo ulteriormente che $q_0 \Rightarrow_G^* w$ se e solo se $q_0 \Rightarrow_G^* wq_j$ e q_j è finale (per la regola di costruzione della grammatica numero 4). Concludiamo che $q_0 \Rightarrow_G^* w$ se e solo se $\delta(q_0, w) \in F$, che equivale a dire: se e solo se $w \in L$.

ÿ

Esempio 4.1
Sia dato il seguente automa, descritto dal diagramma degli stati:



Il linguaggio accettato dall'automa è generabile dalla grammatica con assioma q_0 e dalle seguenti regole di produzione:
$q_0 \rightarrow aq_1$, $q_2 \rightarrow aq_1$, $q_1 \rightarrow aq_2$,
$q_2 \rightarrow \varepsilon$

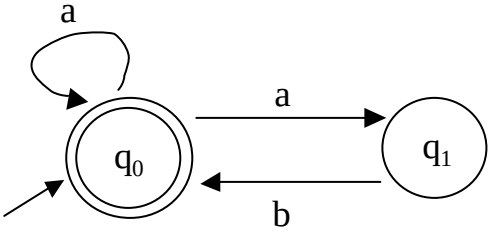
Abbiamo visto che, per ogni automa a stati finiti è possibile costruire una grammatica di tipo 3 che genera il linguaggio riconosciuto dall'automa. Poniamoci il problema inverso: data una grammatica di tipo 3, costruire una automa che riconosce il linguaggio generato dalla grammatica.

Si consideri, a questo riguardo, una grammatica $G =< S,Q,P,S >$ di tipo 3; per la **Proposizione 5.2** in **Cap.1** possiamo supporre, senza perdita di generalità, che la grammatica contenga solo regole del tipo $A \rightarrow \sigma B$, $A \rightarrow \varepsilon$. Cercando di invertire la costruzione data in **Proposizione 4.1**, possiamo costruire un grafo con archi etichettati S come segue:

1. Vertici del grafo sono i metasimboli in Q.
2. Nel grafo c'è un arco da A a B etichettato σ se $A \rightarrow \sigma B$ è una produzione di G.
3. Nodo iniziale è S
4. A è un nodo finale se $A \rightarrow \varepsilon$ è una regola di G.

Esempio 4.2
Data la grammatica $G =< S,Q,P,S >$ con regole di produzione
$q_0 \rightarrow aq_0$
$q_0 \rightarrow aq_1$
$q_1 \rightarrow bq_0$
$q_0 \rightarrow e$

Il grafo con archi etichettati associato è il seguente:



Interpretando i metasimboli come stati, osserviamo che il grafo precedente non è il grafo degli stati di un automa. In un automa, infatti, dato uno stato q e un simbolo σ esiste *un unico stato prossimo* $\delta(\sigma,q)$; nel grafo precedente, ci sono invece due transizioni etichettate con σ che portano da q_0 a due distinti stati q_0 e q_1. Possiamo interpretare questo fatto come una specie di *non determinismo*: se il sistema si trova in un dato stato (ad esempio q_0), l'arrivo di un messaggio non porta necessariamente ad uno stato prossimo univocamente individuato dallo stato presente e dal messaggio, bensì porta in uno tra un insieme di stati *possibili* (nel nostro esempio q_0 o q_1). Questo porta alla seguente definizione:

Definizione 4.1 Un *automa a stati finiti non deterministico* A è un sistema $A = < Q , \Sigma , R , q_o , F >$, dove Q è *un insieme finito di stati,* Σ è un alfabeto finito, $R: Q\times\Sigma\times Q\rightarrow\{0,1\}$ è la *relazione di transizione*, $q_o\in Q$ è lo *stato iniziale,* $F\subseteq Q$ è l'insieme degli *stati finali*.

La funzione $R: Q\times\Sigma\times Q\rightarrow\{0,1\}$ può essere univocamente rappresentata dalla relazione $R'\subseteq Q\times\Sigma\times Q$, dove $(q, \sigma, q') \in R'$ se e solo se $R(q, \sigma, q')=1$. Allo stesso modo, $R: Q\times\Sigma\times Q\rightarrow\{0,1\}$ può essere rappresentata dalla funzione $R'': Q\times\Sigma \rightarrow P(Q)$, dove P(Q) è l'insieme delle parti di Q e $R''(q, \sigma) = \{q' | R(q, \sigma, q')=1\}$.

L'automa a stati finiti nondeterministico è rappresentabile da un grafo etichettato, i cui vertici sono gli stati di q e da q a q' c'è un arco etichettato con σ se e solo se $R(q, \sigma, q')=1$. Il grafo avrà un vertice speciale corrispondente allo stato iniziale e altri vertici opportumametne marcati corrispondenti agli stati finiali.

Una parola $w= x_1 \dots x_m$ è riconosciuta dall'automa non deterministico A se essa induce nel grafo degli stati almeno un cammino $s_0 , s_1, \dots, s_m$ dallo stato iniziale $s_0 = q_o$ a uno stato finale $s_m\in F$, cioè se:

1. $s_0 = q_o$
2. $R(s_0, x_1 , s_1)= \dots = R(s_k, x_{k+1} , s_{k+1})= \dots =R(s_{m-1}, x_m , s_m)=1$
3. $s_m\in F$

Il linguaggio L(A) riconosciuto dall'automa non deterministico A è l'insieme delle parole riconosciute.

Osserviamo che un automa a stati finiti $A = < Q , \Sigma , \delta , q_o , F >$ è un particolare automa non deterministico $A = < Q , \Sigma , R , q_o , F >$, in cui $R(q, \sigma,q')=1$ se e solo se $q'=\delta(q, \sigma)$. La relazione R è equivalente in tal caso ad una funzione $\delta: \Sigma\times Q\rightarrow Q$, ed ogni parola $w= x_1 \dots x_m$ induce un unico cammino partente dallo stato iniziale.

Ovviamente esistono automi a stati finiti non deterministici che non sono automi a stati (si veda **Esempio 4.2**). Ciò nonostante, i linguaggi riconosciuti da automi non deterministici coincidono con quelli riconosciuti da automi deterministici:

Proposizione 4.2 Per ogni linguaggio L riconosciuto da un automa a stati finiti non deterministico finiti esiste un automa a stati finiti che lo riconosce.

Dimostrazione: Sia $A = < Q , \Sigma , R , q_o , F >$ un automa a stati finiti non deterministico che riconosce L. Possiamo costruire il seguente automa deterministico $A' = < Q' , \Sigma , \delta , q_{o'} , F' >$ che riconosce lo stesso linguaggio come segue:

1. Q' = 2^Q, cioè gli stati del nuovo automa A' sono i sottoinsiemi degli stati di A.
2. Se $X \subseteq Q$, allora $\delta(X, \sigma)=\{q' | R(q,\sigma,q')=1 \text{ e } q \in X\}$, cioè $\delta(X, \sigma)$ è l'insieme di stati accessibili da X con un arco etichettato con σ.
3. $q_{o'} = \{ q_o \}$
4. $F' = \{X | X \subseteq Q \text{ e } X \cap F \neq \emptyset\}$, cioè F' è formato dai sottoinsiemi di Q che contengono almeno un stato in F.

Per induzione sulla lunghezza della parola, a causa delle condizioni 2. e 3. di dimostra facilmente che:

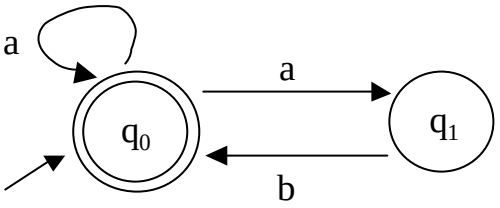
$\delta(q_{o'}, x_1 \dots x_m,) = \{q | x_1 \dots x_m$ induce un cammino $q_o, s_{1,\dots}, s_m$ da q_o allo stato $s_m = q\}$

Per la 4. segue infine che $x_1 \dots x_m$ è riconosciuta da A se e solo se è riconosciuta da A'.

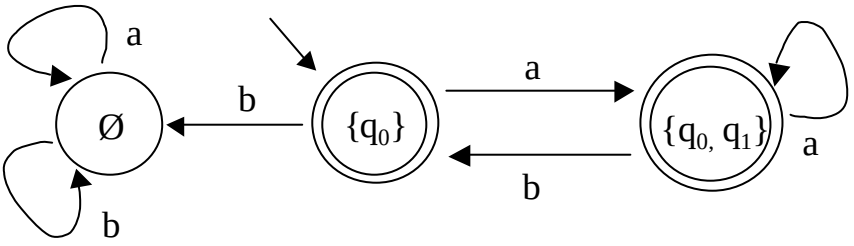
ÿ

Esempio 4.2

Dato l'automa non deterministico descritto dal seguente grafo:



Un automa deterministico equivalente, che riconosce cioè lo stesso linguaggio, ottenuto con la costruzione mostrata in **Proposizione 4.2**, è il seguente:



Nota bene: nel grafo così ottenuto non viene visualizzato lo stato $\{q_1\}$ in quanto tale stato non è osservabile.

Possiamo a questo punto concludere:

Teorema 4.1 Le due seguenti affermazioni sono equivalenti:

(1) L è generato da una grammatica di tipo 3

(2) L è riconosciuto da un automa a stati finiti

Dimostrazione:

(1) implica (2).

E' il contenuto di **Proposizione 4.1.**

(2) implica (1).

Data una grammatica G di tipo 3 che genera L, si costruisce il grafo etichettato associato, che può essere interpretato come automa a stati finiti non deterministico che riconosce L. Si costruisce infine come in **Proposizione 4.2** un automa a stati finiti che riconosce L.

ÿ

In **Proposizione 4.2** abbiamo mostrato che per ogni automa non deterministico a stati finiti esiste un automa deterministico equivalente. Va per contro osservato che, se l'automa non deterministico ha M stati, il nuovo automa deterministico può avere fino a 2^M stati, cosa che rende inutilizzabile la costruzione in molte applicazioni.

5 Automi a stati finiti e espressioni regolari

In questa sezione introduciamo una classe di espressioni (le espressioni regolari) ed associamo in modo naturale ad ogni espressione un linguaggio. Mostriamo poi che la classe di linguaggi denotati da espressioni regolari è la classe di linguaggi riconosciuti da automi a stati finiti (Teorema di Kleene).

Definizione 5.1 Dato un alfabeto Σ, le *espressioni regolari* su Σ sono definite induttivamente:
1. $\emptyset$, ε, σ (per $\sigma \in \Sigma$) sono espressioni regolari
2. se p, q sono espressioni regolari, allora (p+q), (p.q), (p*) sono espressioni regolari

Richiamiamo ora che, dati linguaggi L_1, L_2 e L sull'alfabeto Σ:
1. Unione di L_1 e L_2 è il linguaggio $L_1 \cup L_2 = \{x | x \in L_1 \text{ o } x \in L_2\}$
2. Prodotto di L_1 e L_2 è il linguaggio $L_1 . L_2 = \{xy | x \in L_1 \text{ e } y \in L_2\}$
3. Chiusura (di Kleene) di L è il linguaggio $L^* = L^0 \cup L^1 \cup \ldots \cup L^k \cup \ldots = \bigcup_{k=0}^{\infty} L^k$

Ad ogni espressione regolare p associamo un linguaggio L (e diremo che p *denota* L) come segue:
1. $\emptyset$ denota il linguaggio vuoto, ε denota il linguaggio $\{\varepsilon\}$, σ denota il linguaggio $\{\sigma\}$
2. se p,q,m denotano rispettivamente L_1, L_2 e L, allora (p+q), (pq), (m*) denotano rispettivamente $L_1 \cup L_2$, $L_1 . L_2$, L*.

Esempio 5.1
a*b*a*+(ab)* è una espressione regolare che denota il linguaggio L dove:

$$L = \{w \mid w = a^j b^k a^l \ (j,k,l \geq 0) \text{ o } w = (ab)^k \ (k \geq 0)\}.$$

Il linguaggio $\{w \mid w = a^{2k+1} \ (k \geq 0)\}$ è denotato dall'espressione regolare a.(aa)* , oppure da (aa)*a o (aa)*a(aa)* (o da altre ancora).

Osserviamo che le espressioni regolari denotano linguaggi in modo *composizionale*: una data espressione indica le operazioni di unione, prodotto e chiusura che, applicate ai linguaggi-base $\emptyset$, $\{\varepsilon\}$, $\{\sigma\}$, permettono di ottenere il linguaggio denotato. Questo permette l'uso di tecniche induttive per mostrare proprietà dei linguaggi denotati da espressioni regolari. Ad esempio, per mostrare che una proprietà P vale per tutti i linguaggi denotati da espressioni regolari, basta provare che:
1. I linguaggi base $\emptyset$, $\{\varepsilon\}$, $\{\sigma\}$ verificano la proprietà P
2. Se i linguaggi A,B verificano la proprietà P, allora $A \cup B$, AB, A* verificano la proprietà P.

Esempio 5.2
Data una parola $w = x_1 x_2 .. x_m$, la sua *trasposta* w^R è la parola $w^R = x_m x_{m-1} .. x_1$; dato un linguaggio L, il suo *trasposto* L^R è il linguaggio $\{w \mid w^R \in L\}$. Vogliamo provare che se L è denotato da una espressione regolare, allora anche L^R lo è. A tal riguardo, basta osservare:

1. $\emptyset^R = \emptyset$, $\{\varepsilon\}^R = \{\varepsilon\}$, $\{\sigma\}^R = \{\sigma\}$,

2. $(A \cup B)^R = A^R \cup B^R$, $(AB)^R = B^R A^R$, $(A^*)^R = (A^R)^*$

Le regole precedenti permettono, conoscendo l'espressione che denota L, di costruire l'espressione che denota L^R. Per esempio, se (a+b)(ac+ba)* denota L, allora (ca+ab)*(a+b) denota L^R.

I linguaggi denotati da espressioni regolari sono tutti e soli quelli riconosciuti da automi a stati finiti, come enunciato nel seguente:

Teorema (di Kleene) 5.1 Le due seguenti affermazioni sono equivalenti:
(1) L è denotato da una espressione regolare
(2) L è riconosciuto da un automa a stati finiti
Dimostrazione:
(1) implica (2).
Basta provare che:
1. $\emptyset$, $\{\varepsilon\}$, $\{\sigma\}$ sono riconosciuti da automi a stati finiti
2. Se A, B sono riconosciuti da automi a stati finiti, allora $A \cup B$, AB, A* lo sono.
Per il punto 1., basta osservare che:

sono automi che riconoscono rispettivamente $\emptyset$, $\{\varepsilon\}$, $\{\sigma\}$.
Per il punto 2., supponiamo che A,B siano riconosciuti da automi a stati finiti.
Per la **Proposizione 4.1**, A è generato da una grammatica G'=< S,Q',*P'*,S' > di tipo 3 e B è generato da una grammatica G"=< S,Q",*P"*,S" > di tipo 3. Senza perdita di generalità, possiamo supporre che le regole siano del tipo $q_k \rightarrow \sigma q_j$ oppure $q_k \rightarrow \varepsilon$, ed inoltre che Q' e Q" siano insiemi disgiunti.
Si verifica facilmente che:
a. $A \cup B$ è generato dalla grammatica G_1=< S,Q_1,P_1,S_1 >, dove Q_1 contiene i metasimboli in Q', i metasimboli in Q" e un nuovo metasimbolo S_1, mentre P_1 contiene le regole in *P'*, le regole in *P"* e le nuove regole $S_1 \rightarrow S'$, $S_1 \rightarrow S"$.
b. AB è generato dalla grammatica G_2=< S,Q_2,P_2,S' >, dove Q_2 contiene i metasimboli in Q' e in Q", mentre P_1 contiene le regole in *P'*, ad esclusione di quelle del tipo $q' \rightarrow \varepsilon$, tutte le regole in *P"* e le nuove regole $q' \rightarrow S"$ per ogni metasimbolo q' in Q' per cui $q' \rightarrow \varepsilon$ è una regola in *P'*.
c. A* è generato dalla grammatica G_3=< S,Q', P_3, S' >, dove P_3 contiene le regole in P' più le nuove regole $q' \rightarrow S'$ per ogni metasimbolo q' in Q' per cui $q' \rightarrow \varepsilon$ è una regola in *P'*.
(2) implica (1)
Sia L riconosciuto dall'automa a stati finiti $A = < Q , \Sigma , \delta , q_o , F >$; vogliamo esibire una espressione regolare che denota L.
A tal riguardo, per ogni stato q_k consideriamo l'automa $A_k = < Q , \Sigma , \delta , q_k , F >$ che differisce da A solo per lo stato iniziale, che è q_k anziché q_0. Sia X_k il linguaggio riconosciuto da A_k, così che $L=X_0$.
Osserviamo che:
1. $\varepsilon \in X_k$ se e solo se q_k è stato finale.
2. $\sigma w \in X_k$ se e solo se $\delta(q_k,\sigma)=q_j$ e $w \in X_j$.
Per ogni stato finale q_k , possiamo quindi scrivere l'equazione:

$$X_k = \{\varepsilon\} \cup (\cup_{\sigma \in \Sigma, \delta(qk, \sigma)=qj} \sigma X_j)$$

Analogamente, per ogni stato non finale q_s possiamo scrivere l'equazione:

$$X_s = (\cup_{\sigma \in \Sigma, \delta(\sigma,qs)=qj} \sigma X_j)$$

Abbiamo ora un sistema in cui le incognite sono linguaggi; il numero di incognite è inoltre pari al numero delle equazioni. Va segnalato inoltre che il sistema è "lineare a destra" (la parte sinistra di ogni equazione è una variabile, mentre la parte destra è una unione di prodotti, ed in ogni prodotto l'incognita compare al più una volta ed in ultima posizione). Allo scopo di risolvere tale sistema, cominciamo con lo studio dell'equazione

$$(\#) \quad X=AX \cup B$$

dove A,B sono linguaggi e supponiamo che $\varepsilon \notin A$. Tale equazione ammette una sola soluzione che è $X=A^*B$

- A^*B è una soluzione, come si verifica immediatamente:
  $A(A^*B) \cup B=A(\{\varepsilon\} \cup A \cup A^2 ... \cup A^k ..)B \cup B=(A \cup A^2 ... \cup A^k ..)B \cup B=(A \cup A^2 ... \cup A^k .. \cup \{\varepsilon\})B=A^*B$
- Tale soluzione è inoltre la sola. Supponiamo infatti che ci siano due soluzioni distinte X,Y tali che $X=AX \cup B$ e $Y=AY \cup B$. Sia h la lunghezza della più corta parola che distingue X da Y (si trova in un linguaggio e non nell'altro) e sia inoltre s la lunghezza della più corta parola in A ($s>0$ poiché per ipotesi $\varepsilon \notin A$). La più corta parola che distingue $AX \cup B$ da $AY \cup B$ risulta allora di lunghezza $h+s>h$: assurdo, poiché $X=AX \cup B$ e $Y=AY \cup B$.

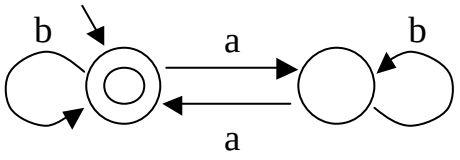
Il sistema può essere risolto per sostituzione: si fissa una equazione e una incognita in essa contenuta, la si risolve come (#) e si sostituisce il risultato in ogni altra equazione, applicando poi opportunamente la proprietà distributiva. Si ottiene un sistema dello stesso tipo, con una equazione in meno ed una incognita in meno.

Al termine ogni linguaggio X_k, e quindi in particolare $L= X_0$, verrà ottenuto applicando a Ø, $\{\varepsilon\}$, $\{\sigma\}$ un numero finito di volte le operazioni di unione, prodotto e chiusura.

ÿ

Esempio 5.3

Dato il seguente automa che riconosce il linguaggio L, trovare l'espressione regolare che denota L.



Detto q_0 lo stato iniziale e q_1 l'altro stato, otteniamo il sistema (utilizzando per comodità il simbolo + invece di $\cup$):

$X_0=\varepsilon + aX_1+bX_0$

$X_1= aX_0+bX_1$

Risolvendo la seconda equazione:

$X_1= b^*aX_0$

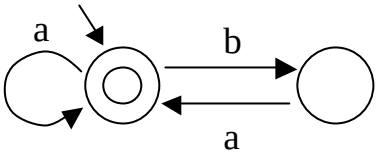
Sostituendo la soluzione data dalla seconda equazione nella prima e raccogliendo:

$X_0 = \varepsilon +a(b^*aX_0)+ bX_0= \varepsilon + ab^*aX_0+ bX_0= \varepsilon + (ab^*a + b) X_0$

Risolvendo:

$X_0 = (ab^*a +b)^* \varepsilon=(ab^*a +b)^*=L$

Esempio 5.4
Il linguaggio L={w | w∈{a,b}* e bb non è fattore di w} è riconosciuto dal seguente automa non deterministico:



Trovare l'espressione regolare che determina lo stesso linguaggio.
Detto q_0 lo stato iniziale e q_1 l'altro stato, otteniamo il sistema (utilizzando come prima il simbolo + invece di ∪):

$X_0 = aX_0+bX_1+\varepsilon$
$X_1 = aX_0$

Sostituendo la soluzione data dalla seconda equazione nella prima e raccogliendo:

$X_0 = (a+ba)X_0+\varepsilon$

Risolvendo:

$X_0 = (a+ba)^*\varepsilon$

Da cui L= X_0 = (a+ba)*, ottenendo una espressione regolare che denota L.

Una delle implicazioni del teorema di Kleene è la chiusura dei linguaggi regolari sotto unione, prodotto e chiusura di Kleene: se A e B sono linguaggi regolari, allora A∪B, AB, A* lo sono.
E' ulteriormente possibile provare che i linguaggi regolari sono chiusi rispetto alle operazioni di complemento e intersezione:

Teorema 5.2 Se A, B sono linguaggi regolari, allora A^c e A∪B lo sono.
Dimostrazione: Se A è regolare allora è riconosciuto da un automa a stati finiti deterministico < Q , Σ , δ , q_0 , F >. E' immediato verificare che l'automa < Q , Σ , δ , q_0 , Q\F >, identico al precedente salvo l'aver scambiato gli stati non finali con gli stati finali, riconosce A^c . Quindi i linguaggi regolari sono chiusi rispetto al complemento. Essendo chiusi rispetto all'unione, sono allora chiusi rispetto all'intersezione, poiché per la legge di De Morgan:

$$A \cup B = (A^c \cup B^c)^c$$

ÿ

6 Espressioni regolari in UNIX

Le espressioni regolari sono molto usate in UNIX come formalismo per specificare linguaggi sull'alfabeto dei caratteri. Molti comandi importanti effettuano infatti elaborazioni su testi e possono essere applicati a linguaggi specificati attraverso sottoclassi di espressioni regolari (oppure, in certi casi, estensioni delle espressioni regolari). Tra essi citiamo awk, ed, ex, grep, pg, sed, lex e vi. Presentiamo qui due esempi:

- le espressioni regolari semplici e il comando grep;
- le espressioni regolari estese e il comando lex.

6.1 Espressioni regolari semplici

Abbiamo visto che, dato un alfabeto Σ, le espressioni regolari su Σ sono opportune parole sull'alfabeto Σ esteso coi simboli Ø, ε, + , . , *, (,) . Tali parole sono ottenute a partire dai simboli Ø, ε, σ (per $\sigma \in \Sigma$) applicando iterativamente la seguente regola: se p, q sono espressioni regolari, allora anche (p+q), (p.q), (p)* lo sono. Sottoclassi di espressioni regolari possono essere ottenute ponendo *restrizioni* alla precedente regola di formazione, togliendo eventualmente le parentesi se il linguaggio denotato dall'espressione non cambia (ad esempio, ((a+b)+c) può essere riscritto come (a+b+c)). Vediamo alcuni esempi:

Dato un alfabeto Σ, le *classi* su Σ sono espressioni regolari del tipo $(x_1 + x_2 + \ldots + x_n)$, dove x_k è una lettera in Σ. Ad esempio, se Σ = {a,b,c,d}, allora (a+b+c+a) oppure (b + d) oppure (c) sono classi, mentre a* oppure acd oppure (a+c)*bd non lo sono. Il linguaggio denotato da una classe è un sottoinsieme di Σ.

Le *espressioni regolari semplici* sono espressioni regolari del tipo $(c_1.c_2 \ldots .c_n)$, dove c_k è o una classe, o una classe seguita da *. Ad esempio ((a+d).(b+c)*.(a+d)) è una espressione regolare semplice, che denota il linguaggio di parole che iniziano col simbolo a oppure d, seguito da una parola sull'alfabeto {b,c} e terminano col simbolo a oppure d. (a*+b*) è invece una espressione regolare ma non è una espressione regolare semplice.

Essenzialmente, le *espressioni regolari semplici in UNIX* sono espressioni regolari semplici sull'alfabeto dei caratteri ASCII, con due avvertimenti:

- alcuni caratteri ASCII sono usati in UNIX per denotare operazioni, quindi non possono essere usati direttamente come caratteri; tali caratteri potranno tuttavia essere codificati con opportune parole.
- Poiché i caratteri ASCII sono 256, si sono inventati trucchi per specificare alcuni sottoinsiemi di caratteri in maniera compatta.

Per quanto riguarda il primo punto, escluderemo i caratteri ^, $, ., \ , - , [,] , * , + , ?, {, }, |, (,), ", /, %, <, >, che denotano operazioni in UNIX e ci limiteremo a considerare i rimanenti caratteri. Indicheremo con Σ i rimanenti caratteri, che vengono ordinati con un ordine totale:

$$\Sigma = \{\ldots<0<1<\ldots<9<\ldots<A<B<\ldots<Z<\ldots<a<b<\ldots<z<\ldots\}$$

Un *intervallo* dell'ordine totale può essere identificato dai suoi estremi: con x-y denotiamo quindi tutti i caratteri z tali che $x \le z \le y$. L'insieme degli intervalli è allora il linguaggio $I(\Sigma) = \Sigma\text{-}\Sigma = \{x\text{-}y \mid x,y \in \Sigma\}$; si osservi che $\Sigma \cup I(\Sigma)$ forma un codice.

Le *classi di caratteri* in UNIX sono le parole nel linguaggio $[((\Sigma \cup I(\Sigma))^*] \cup [\hat{}(\Sigma \cup I(\Sigma))^*]$, e cioè le parole del tipo $[x_1\ x_2\ \ldots\ x_n]$ o $[\hat{}\ x_1\ x_2\ \ldots\ x_n]$ con $x_k \in \Sigma \cup I(\Sigma)$. Ogni classe w di caratteri viene interpretata come il sottoinsieme $K(w) \subseteq \Sigma$, definito come segue:

- se $x, y \in \Sigma$ allora $K([x]) = \{x\}$, mentre $K([x\text{-}y]) = \{z \mid x \le z \le y , z \in \Sigma\}$
- $K([x_1\ x_2\ \ldots\ x_n]) = K([x_1]) \cup K([x_2]) \cup \ldots \cup K([x_n])$
- $K([\hat{}x_1\ x_2\ \ldots\ x_n]) = \Sigma \setminus K([x_1\ x_2\ \ldots\ x_n])$

Esempio 6.1
K([A]) = {A}, K([aAb]) = {a,b,A}, K([3-7]) = {3,4,5,6,7}, K([13-79]) = K([1]) ∪ K([3-7]) ∪ K([9])
= {1,3,4,5,6,7,9}, K([^3-5A-D]) = Σ \ {3,4,5,A,B,C,D}= {.., 0,1,2,6,7,..,9, …, E,F, .., Z,…,a,b,..,z, …}.

Le *espressioni regolari semplici in UNIX* sono espressioni regolari del tipo $c_1 c_2 \ldots c_n$, dove c_k è o una *classi di caratteri*, o una *classi di caratteri* seguita da * . Interpretando come prima le classi di caratteri come sottoinsiemi di Σ, il simbolo * come chiusura di Kleene e la sequenza come prodotto di linguaggi, ogni espressione regolare semplice è interpretata come linguaggio contenuto in Σ*.

Esempio 6.2
[1-3][bd] vale il linguaggio {1,2,3}·{b,d}= {1b,1d,2b,2d,3b,3d}
[1-3]*[bd] vale il linguaggio {1,2,3}*·{b,d}, cioè le parole sull'alfabeto {1,2,3} seguite da b oppure d.

6.2 String matching in UNIX: il comando grep

Un importante problema che si ritrova frequentemente nella elaborazione di testi è lo "string matching". Nella versione più semplice, tale problema richiede, date due parole p,t∈Σ* dette rispettivamente pattern e testo, di determinare se il pattern è un fattore del testo, cioè se t=xpy per opportuni x,y. Tale problema può essere esteso considerando un linguaggio L come pattern e chiedendo in quali posizioni di t compaiono parole di L.

Il comando grep (Global Regular Expression Printer) risolve in UNIX un problema di string matching in cui il linguaggio pattern viene specificato da una espressione regolare semplice e il testo è un file di testo ASCII. Per esempio, per ricercare la parola "automa" in un file di testo UNIX chiamato dispensa.txt basterà digitare il comando:

grep 'automa' dispensa.txt

In uscita verranno visualizzate tutte le righe del file dispensa.txt contenenti la parola "automa".

6.3 Espressioni regolari estese

Essenzialmente, le espressioni regolari semplici sono concatenazioni di classi o classi iterate. Partendo dalle espressioni regolari semplici, si possono ottenere le *espressioni regolari estese* applicando iterativamente la regola: se r, s sono *espressioni regolari estese*, allora (rs), (r|s), (r)*, (r)+, (r)? sono *espressioni regolari estese*.
Ad esempio, ([a-d1]2)*|[f]? è una espressione regolare estesa.

Interpretando opportunamente la concatenazione e | come operazioni binarie su linguaggi, *,+,? come operazioni unarie sui linguaggi, ogni espressione regolare estesa denota un linguaggio sull'alfabeto Σ dei caratteri. In particolare, se X e Y sono linguaggi su Σ, allora:

- $XY = \{xy | x \in X, y \in Y\}$ (Usuale prodotto di linguaggi)
- $X|Y = X \cup Y$ (Usuale unione di linguaggi)
- $X^* = \{\varepsilon\} \cup X \cup X^2 \cup \ldots \cup X^k \cup \ldots$ (Usuale chiusura di Kleene)
- $X+ = X \cup X^2 \cup \ldots \cup X^k \cup \ldots$ (in precedenza denotato con X^+)
- $X? = \{\varepsilon\} \cup X$

Ad esempio, usando la precedente denotazione, l'espressione regolare estesa , ([a-d1]2)*|[f]? denota il linguaggio {a2,b2,c2,d2,12}*∪{ε}∪{f}.

6.4 Analizzatore lessicale in UNIX: il comando lex

Le frasi dell'italiano scritto sono potenzialmente infinite, ma costituite da parole, normalmente di poche lettere, riconoscibili perché "separate" tra loro. Queste parole formano il *lessico*, e l'analisi grammaticale le divide in gruppi "logicamente omogenei": i nomi, gli articoli, i verbi e così via.

Esempio 6.3
"sul-castello-di-Verona⊥batte-il-sole-a-mezzogiorno" è una frase, frammento di una nota poesia di Carducci, vista come parola contenente i simboli speciali -,⊥. Nella frase sono facilmente individuabili le parole del lessico: sul, castello, …., mezzogiorno.

L'analisi del lessico non è in grado, ovviamente, di attribuire significato alla frase, pur potendo costituire una utilissima pre-elaborazione per la comprensione. Una situazione analoga si riscontra in linguaggi artificiali quali i linguaggi di programmazione. Il *significato* di un programma, scritto in un dato linguaggio di programmazione, viene attribuito operazionalmente dal *compilatore*. Il compilatore è un algoritmo che, ricevendo in ingresso il testo T di un programma scritto in un dato linguaggio di programmazione (detto *linguaggio sorgente*) lo traduce in un corrispondente programma Z scritto in un linguaggio (detto *linguaggio oggetto*) direttamente eseguibile dall'elaboratore.

La prima fase del processo di compilazione è la cosiddetta *analisi lessicale*. Ogni linguaggio di programmazione ha un suo lessico. Gli elementi del lessico sono divisi in gruppi "logicamente omogenei" talvolta detti *token*. Essi sono ad esempio gli identificatori di variabili (come PIPPO, delta,x, ….), le costanti (come 100, 3.14 , ….), gli operatori (come +, * , -, : , =, ^ , ….), i commenti (seguiti e preceduti da opportuni segnalatori), le parole chiave del linguaggio (if, do,while, …) e così via.

I linguaggi corrispondenti ai token sono generalmente molto semplici, denotabili con espressioni regolari; di conseguenza l'individuazione di prefissati token in un testo T richiede:

1. la costruzione degli automi a stati finiti che riconoscono i linguaggi corrispondenti ai token; tali automi esistono grazie al teorema di Kleene;
2. la scansione del testo T da parte degli automi precedentemente costruiti.

Ogniqualvolta una sottostringa del testo T viene riconosciuta appartenente a un token, l'analizzatore lessicale attiva delle *azioni*, che comportano la sostituzione della sottostringa con una nuova stringa. Alla fine otteniamo una nuova *rappresentazione interna* V del testo T, utile per la compilazione.

Il comando lex di Unix facilita la costruzione di analizzatori lessicali implementando automaticamente i punti 1. e 2. sopra riportati. Il programmatore deve solo definire i token che vuole riconoscere (scrivendo l'espressione regolare estesa che denota il token) e le azioni da compiere ogni qual volta un certo token viene riconosciuto (implementando tali azioni con semplici istruzioni nel linguaggio C).

Diamo un esempio, semplificando a scopo didattico. Il seguente file "tokens" specifica le espressioni regolari estese che denotano i token; di fianco ad ogni espressione viene indicata l'azione da intraprendere, scritta in C:

```
%%
[ \t\n]                         ;
\{                              printf("BEGIN\n");
\}                              printf("END\n");
[()]                            printf("%s\n",yytext);
"//"[^\n]*[\n]                  printf("rem: //\n");
if|else                         printf("key: %s\n",yytext);
[A-Za-z][A-Za-z0-9]*            printf("var: %s\n",yytext);
==|\>|\<|\<=|\>=|!=             printf("opc: %s\n",yytext);
[0-9]+(\.[0-9]+)?               printf("num: %s\n",yytext);
[A-Za-z]+\([A-Za-z"]*\);?       printf("fun: %s\n",yytext);
%%
```

La variabile "yytext" contiene la sottostringa di testo appena riconosciuta.
Ad esempio, consideriamo il testo T = …. ORDINA(pippo) … . Non appena l'analizzatore lessicale legge la stringa ORDINA(pippo) l'automa corrispondente all'espressione `[A-Za-z]+\([A-Za-z"]*\);?` riconosce tale stringa che viene posta in yytext. Viene infine eseguita l'azione `printf("fun: %s\n",yytext);` con l'effetto di sostituire ORDINA(pippo) con fun: ORDINA(pippo).

Il comando:

lex tokens

genera il programma C "lex.yy.c", che corrisponde all'analizzatore lessicale da noi desiderato.
Digitando ora il comando:

cc lex.yy.c -ll -o lexical

compiliamo il programma C "lex.yy.c", ottenendo il file eseguibile "lexical".
Per vedere l'effetto dell'applicazione di "lexical" a un testo, consideriamo come testo il seguente file "prog.c" contenente un semplice programma C:

```
//frammento di un programma C
main()
{
if (budget>=1999.99)
        stampa("ok");
else
        stampa("ko");
}
```

Mandiamo in esecuzione "lexical" sul testo "prog.c" mediante il comando:

lexical<prog.c

A questo punto l'uscita a video sarà:

```
rem: //
fun: main()
BEGIN
key: if
(
var: budget
opc: >=
num: 1999.99
)
fun: stampa("ok");
key: else
fun: stampa("ko");
END
```